RAIRO Operations Research
Will be set by the publisher
DOI: (will be inserted later)

DES EXPLICATIONS POUR RECONNAÎTRE ET EXPLOITER LES STRUCTURES CACHÉES D'UN PROBLÈME COMBINATOIRE

Hadrien Cambazard[1] et Narendra Jussien[1]

Abstract. Identifying structure in a given combinatorial problem is often a key step for designing efficient search heuristics or for understanding the inherent complexity of the problem. Several Operations Research approaches apply decomposition or relaxation strategies upon such a structure identified within a given problem. The next step is to design algorithms that adaptively integrate that kind of information during search. We claim in this paper, inspired by previous work on impact-based search strategies for constraint programming, that using an explanation-based constraint solver may lead to collect invaluable information on the intimate dynamic and static structure of a problem instance.

Résumé. L'identification de structures propres à un problème est souvent une étape clef pour la conception d'heuristiques de recherche comme pour la compréhension de la complexité du problème. De nombreuses approches en Recherche Opérationnelle emploient des stratégies de relaxation ou de décomposition dès lors que certaines structures idoines ont été identifiées. L'étape suivante est la conception d'algorithmes de résolution qui puissent intégrer à la volée, pendant la résolution, ce type d'information. Cet article propose d'utiliser un solveur de contraintes à base d'explications pour collecter une information pertinente sur les structures dynamiques et statiques inhérentes au problème.

Received November 16, 2006. Accepted November 20, 2006.

[1] École des Mines de Nantes – LINA CNRS FRE 2729, 4 rue Alfred Kastler, BP 20722, 44307 Nantes Cedex 3, France ; Hadrien.Cambazard@emn.fr,Narendra.Jussien@emn.fr

© EDP Sciences 2006

1. Introduction

Les stratégies de recherche générique pour résoudre des problèmes d'optimisation combinatoires constituent le Saint-Graal pour les communautés Recherche Opérationnelle (RO) et PPC (Programmation par Contraintes). Différentes pistes sont explorées à l'heure actuelle : adapter dynamiquement la manière dont le solveur s'emploie à résoudre un problème, identifier des structures spécifiques à une instance particulière pour accélérer la recherche, etc. Dans tous les cas, la clef est de pouvoir identifier, comprendre et utiliser les structures intimes présentes dans les instances d'un problème combinatoire donné [12, 25, 26]. Par le passé, Bessière [3] a déjà cherché à prendre en compte le voisinage des variables. Refalo [23] a récemment introduit des stratégies d'impact pour la programmation par contraintes qui s'appuient sur la propagation et prend ainsi en compte des structures du problème. Dans cet article, nous nous intéressons aux relations entretenues par les variables du problème. L'objectif est d'identifier et de différencier à la fois les structures dynamiques (créées par l'algorithme de recherche) et statiques (relatives à une instance) du problème. Une structure est entendue comme un sous-ensemble de variables qui joue un rôle spécifique au sein du problème. Nous définissons à cette fin, plusieurs mesures d'impacts ainsi que le graphe de relations leur correspondant de manière à pouvoir :

- reconnaître les structures cachées (invisibles dans le réseau de contraintes) ;
- concevoir des stratégies de recherche génériques ;
- examiner l'utilisation potentielle de l'analyse des impacts pour des méthodes procédant par décomposition, telle que la décomposition de Benders.

Nos nouvelles mesures d'impacts s'appuient sur le calcul d'explications qui fournissent une information sur l'enchaînement des inférences effectuées au cours de la propagation et donc implicitement des relations au sein de l'ensemble des variables.

L'article est organisé de la manière suivante : la section 2 introduit les bases et les motivations de notre travail ; différentes mesures d'impact et leur graphe associé sont présentés en section 3 en distinguant leur capacité respective à refléter des structures dynamiques et statiques sur un exemple concret. Finalement, nous essayons d'évaluer leur intérêt à la fois comme heuristique de recherche mais aussi pour la conception d'algorithmes de résolution dédiés inspirés de la décomposition logique de Benders.

2. Les stratégies de recherche pour des problèmes structurés

Les stratégies de recherche efficaces exploitent certains aspects ou caractéristiques spécifiques d'un problème (ou d'une instance) donnée. En Recherche Opérationnelle, les stratégies de relaxations ou décompositions exploitent le fait

qu'un problème sous-jacent (ou une partie du problème originel) peut être traité comme un problème classique (flot compatible ou maximum, plus courts chemins, sac à dos, etc.). Cet aspect du problème est souvent désigné sous le terme de *structure*.

Un problème est souvent dit structuré si ses composantes (variables et/ou contraintes) ne jouent pas toutes le même rôle, ou ne revêtent pas la même importance au sein du problème. La complexité d'un tel problème réside souvent dans l'interaction et l'influence mutuelle de ses différentes composantes. Une difficulté centrale dans l'identification de telles structures tient au fait qu'une structure n'existe pas nécessairement d'un point de vue statique (au niveau de l'instance). C'est souvent l'interaction entre une instance particulière et les choix de l'algorithme de recherche lui-même qui engendre une telle structure. Elle est désignée par la suite sous le nom de *structure dynamique* et correspond par exemple à une série de choix initiaux structurants ou à la création de nouvelles relations entre les variables dues à l'ajout de nouvelles contraintes lors de la recherche.

Différentes notions ont été introduites pour caractériser les structures, les *backbones* [20] désignent par exemple les affectations qui font partie de toutes les solutions. Les *backdoors*, récemment introduits en [26] sont un concept intéressant pour caractériser des structures cachées au sein d'un problème. Ils sont définis d'un point de vue informel comme des sous-ensembles de variables qui encapsulent toute la combinatoire du problème : une fois cet ensemble instancié, le sous-problème restant peut être résolu très rapidement. Différentes stratégies de recherche s'appuient sur le principe des *backdoors*. Les deux suivantes sont au cœur de notre démarche :

- les heuristiques de branchement en programmation par contraintes s'efforcent d'orienter la recherche le plus tôt possible vers des variables constituant un *backdoor* dans la mesure où les choix ont pour but de simplifier au maximum le problème. Elles sont basées sur une idée simple : sélectionner une variable qui contraint au maximum le problème et mène sur un espace de recherche aussi petit que possible. Ce principe (couramment désigné par le terme de *first-fail*) est souvent mis en œuvre en prenant en compte le degré et le domaine courant des variables (voir [5], [3] pour des variantes plus pertinentes). Plus récemment, Refalo [23], a proposé de caractériser l'impact d'une décision en examinant en moyenne la réduction de l'espace de recherche engendrée par cette décision (une autre manière d'identifier des *backdoors*).
- la décomposition de Benders [2] s'apparente également à une stratégie s'appuyant sur l'existence de *backdoors*. C'est une stratégie de résolution basée sur une partition des variables du problème en deux ensembles x et y. Un problème maître fournit une affectation réalisable x^* et un sous-problème essaye de compléter cette affectation sur les variables y. Si cela n'est pas possible, le sous-problème produit alors une coupe (une contrainte) ajoutée au problème maître de façon à éliminer l'affectation x^* courante. Les coupes intéressantes sont celles qui permettent d'éliminer non seulement x^* mais également toutes les classes d'affectation partageant avec x^*

les caractéristiques qui rendent x^* infaisable ou sous-optimal. Cette technique est dédiée à des problèmes qui présentent des structures spécifiques. Le problème maître est basé sur un ensemble pertinent de variables qui vérifie en général les hypothèses suivantes :

(1) le sous-problème résultant est simple. En pratique, plusieurs sous-problèmes de petites tailles sont utilisés, simplifiant la recherche souvent exhaustive nécessaire à la production de la coupe ;
(2) la coupe de Benders est suffisamment pertinente pour garantir une convergence rapide de l'ensemble de la technique.

Dans une telle décomposition, les variables du problème maître constituent un *backdoor*. En effet, l'hypothèse (1) nous assure que le sous-problème peut être résolu efficacement une fois le maître instancié. Par ailleurs, si le sous-problème résultant peut être résolu de manière polynomiale (il est dans ce cas analogue à un *strong backdoor*), une coupe plus puissante correspondant au conflit minimal peut souvent être extraite.

Pour la dernière technique abordée, les structures demandent à être identifiées avant que la recherche ne commence. D'ordinaire, c'est l'analyse du réseau de contraintes qui permet d'extraire certaines caractéristiques intéressantes d'une instance. Par exemple, il est classique en coloration de graphes de commencer par rechercher les cliques de taille maximum pour fournir à la fois des bornes inférieures et éventuellement renforcer la propagation en ajoutant des contraintes globales du type *all-different* [24]. Une telle analyse ne procure qu'une information sur les structures statiques. Néanmoins, des structures statiques cachées ou dynamiques sont d'un grand intérêt pour de nombreuses stratégies de recherche. Naturellement leur identification est au moins aussi coûteuse que la résolution du problème d'origine. Nous pensons que, dans le cadre de la programmation par contraintes, la propagation fournit une information qui devrait permettre d'approximer de telles structures. Nous pensons que le calcul des explications (trace limitée, lisible et exploitable de la propagation) est une manière d'extraire cette information de la phase de propagation.

3. Des explications pour l'identification de structures

Refalo [23] a introduit une mesure d'impacts dans l'objectif de détecter les choix qui engendrent la plus grande réduction de l'espace de recherche. Il propose donc de caractériser l'impact d'une décision en comparant le produit cartésien des domaines avant et après la décision en question. Nous chercherons à aller plus loin en analysant où cette réduction intervient et dans quelle mesure les choix antérieurs sont impliqués. Nous étendrons ces mesures à un graphe d'impact entre variables, prenant à la fois en compte l'effet des décisions passées et la réelle implication d'un choix dans chaque inférence effectuée pendant la résolution.

Notre objectif est d'identifier des variables qui contraignent au maximum le problème, des sous-ensembles de variables particuliers qui entretiennent de fortes

relations ou qui ont un impact important sur l'ensemble du problème (analogue à un *backdoor*). Nous avons axé notre étude sur les points suivants :

- l'impact ou l'influence d'une variable sur la réduction directe de l'espace de recherche ;
- l'impact d'une variable au sein d'une chaîne de déductions même longtemps après que cette variable ait été instanciée ;
- la région de l'espace de recherche sur laquelle s'exerce l'influence d'une variable ainsi que les relations précises entre variables ;

Les explications pour la programmation par contraintes semblent pertinentes pour fournir de telles informations.

3.1. Explications et programmation par contraintes

La programmation par contraintes est un paradigme de résolution générique qui s'appuie sur une exploration arborescente de l'espace couplée à des mécanismes d'inférences (algorithmes de filtrage) destinés à réduire autant que possible l'espace de recherche. On peut voir la recherche comme une procédure d'ajout dynamique de contraintes auxquelles on se référera sous le terme de *contraintes de décision*. On considérera par la suite des contraintes de décision du type affectation : $x_i = a$ où la décision revient à choisir une valeur pour une variable. Chaque décision est propagée à l'ensemble du réseau de contraintes jusqu'à l'obtention d'un point fixe, d'une solution ou d'une contradiction. Dans ce dernier cas, l'algorithme revient sur la dernière décision prise et choisit son alternative. Les explications ont été introduites initialement pour améliorer les algorithmes de recherche arborescente basés sur le retour-arrière (*backtrack*). Elles ont été utilisées plus récemment pour d'autres objectifs comprenant notamment la résolution dynamique de problèmes et l'interaction avec l'utilisateur [16].

Définition 1. Une explication enregistre toute l'information nécessaire pour justifier le comportement du solveur (une réduction de domaine ou une contradiction). Elle est constituée d'un ensemble de contraintes C' (sous-ensemble des contraintes originelles C) et d'un ensemble de décisions dc_1, dc_2, ... prises durant la recherche. L'explication du retrait de la valeur a de la variable v s'écrit ainsi :

$$C' \wedge dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \Rightarrow v \neq a$$

Les explications calculées par le solveur représentent la chaîne logique des inférences consécutives faites par le solveur au cours de la propagation. D'une certaine manière, elles fournissent une trace du comportement du solveur car toutes les opérations se doivent d'être *expliquées*. Par la suite, on notera E l'ensemble des explications calculées depuis le début de la recherche et E_i^{val} l'ensemble des explications calculées pour tous les retraits de la valeur val du domaine de la variable i.

Les explications sont calculées à la volée par chaque contrainte et stockées au niveau des variables. Elles induisent donc une complexité temporelle supplémentaire (les algorithmes de filtrage se voient associés à un algorithme explicatif) mais aussi

spatiale (en $O(nd)$ où n est le nombre de variables et d la taille maximum des domaines car il faut stocker une explication par valeur[1]). On désignera par $|e|$ la taille d'une explication e soit le nombre de contraintes de décisions contenues dans e. Ainsi une explication e_1 est dite plus précise que e_2 si $|e_1| < |e_2|$. En effet, plus une explication est petite, plus elle est précise car plus le nombre d'hypothèses nécessaires à la déduction du retrait de la valeur associée est réduit. Enfin l'âge a_e^d d'une décision d au moment du calcul de l'explication e est défini comme le nombre de décisions prises depuis d au moment où e est produite.

3.2. Caractériser l'impact

Refalo [23] propose de caractériser l'impact d'une décision $x_i = a$, en accord avec le principe du *first fail*, à travers la réduction moyenne de l'espace de recherche engendrée par cette décision. Néanmoins, cette réduction n'intervient pas uniquement au moment où la décision est imposée au problème mais aussi quand d'autres déductions (futures) qui sont en partie basées sur l'hypothèse $x_i = a$ sont faites. L'utilisation d'explications peut fournir une information supplémentaire sur l'implication réelle d'une décision dans un retrait. Une décision passée $x_i = a$ a un impact effectif (du point de vue du solveur) sur une valeur val d'une variable x_j si elle apparaît dans l'explication justifiant ce retrait. On note $I_\alpha(x_i = a, x_j, val)$, l'impact de la décision $x_i = a$ sur la valeur val du domaine de x_j. α est l'index utilisé pour distinguer les différentes mesures.

Notre première approche pour mesurer cet impact consiste à considérer le nombre de fois qu'une décision apparaît dans les explications calculées pour la valeur val de x_j en prenant également en compte la taille de l'explication. En effet, lorsqu'une explication est de petite taille (plus précise), les pré-conditions nécessaires à l'établissement du retrait de valeur associé sont peu nombreuses (et donc plus facilement vérifiables). Le retrait a alors plus de possibilités d'être retrouvé dans la suite de la recherche dans des situations similaires. Ainsi, les relations entre variables doivent être plus fortes. I_0 propose de quantifier cette relation d'influence :

$$I_0(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} 1/\left|e\right|. \tag{1}$$

Différentes mesures basées tout d'abord sur l'activité du solveur et le calcul des explications (mesures I_1 et I_2) sont introduites à partir de cette mesure initiale, avec l'objectif de rendre compte des structures dynamiques. Dans un second temps, une mesure s'appuyant plus sur la réduction de l'espace de recherche (I_3) est proposée pour capturer les structures statiques et aider à guider la recherche.

Comme la recherche oriente à l'évidence la propagation (et *vice-versa*), il semble assez naturel de normaliser ces mesures par rapport à la recherche.

– L'impact est ici normalisé par rapport au nombre de fois $|x_i = a|$ où une décision $x_i = a$ est prise. Le but étant simplement de ne pas surestimer

[1] On ne stocke pas plus d'une explication par valeur pour limiter l'occupation mémoire.

l'impact des décisions prisent fréquemment :

$$I_1(x_i = a, x_j, val) = \frac{I_0(x_i = a, x_j, val)}{|x_i = a|}$$

– Une autre manière de normaliser est de considérer l'âge a_e^d d'une décision d au moment du calcul de l'explication e avec l'objectif de faire décroître l'impact des vieilles décisions. On obtient :

$$I_2(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} \frac{1}{|e| \times a_e^{x_i = a}}.$$

– Le calcul des impacts est dispersé au sein du processus de résolution à chaque calcul d'une explication. C'est une approche sensiblement différente de [23] qui analyse chaque décision séparément pour évaluer son impact instantané. I_3 essaie d'identifier les réductions récurrentes de l'espace de recherche liées à une décision :

$$I_3(x_i = a, x_j, val) = \frac{I_0(x_i = a, x_j, val)}{|\{x_i = a \text{ active } \wedge val \in Dom(x_j)\}|}.$$

$I_3(x_i = a, x_j, val)$ peut être considérée comme la probabilité que la valeur val de x_j soit éliminée si la décision $x_i = a$ est prise. Une telle mesure est mise à jour chaque fois qu'un nouveau retrait intervient dès lors que $x_i = a$ est active. Elle prend en compte la fréquence comme la proportion avec lesquelles une décision est impliquée dans une explication de retrait. Enfin, I_1 met en valeur les vieilles décisions et I_2 les plus récentes. La Section 3.4 montre l'intérêt de ces différents paramètres pour un utilisateur et sa compréhension du comportement du solver.

Ces mesures d'impacts restent fortement dépendantes de l'exploration effectuée et des techniques semblables à celles présentées dans [23] seront employées afin de les initialiser (par la propagation de chaque valeur de chaque variable) et de les affiner (utilisation d'une procédure de *restart* qui consiste à recommencer la recherche pour profiter dès le nœud racine des impacts obtenus par la résolution précédente).

3.3. Relations entretenues par les variables

L'objectif est d'obtenir une représentation synthétique de la structure du problème qui permette à un utilisateur d'analyser et comprendre le problème et le comportement du solveur pendant la résolution. Nous cherchons à exhiber les relations entretenues entre les variables. À cette fin, les mesures d'impacts introduites précédemment sont agrégées sur l'ensemble des valeurs du domaine :

$$\forall \alpha \in [0, 1, 2], \qquad I_\alpha(x_i = a, x_j) = \sum_{val \in D(x_j)} I_\alpha(x_i = a, x_j, val).$$

Un cas particulier se présente pour I_3 où il s'agit de relier l'impact à la réduction de l'espace de recherche engendrée par une variable sur une autre. On considère

dès lors la taille du domaine initial :

$$I_3(x_i = a, x_j) = \frac{|D(x_j)| - \sum_{val \in D(x_j)} (1 - I_3(x_i = a, x_j, val))}{|D(x_j)|}.$$

Dans ce contexte, $1 - I_3(x_i = a, x_j, val)$ correspond en quelque sorte à la probabilité de présence de la valeur val de la variable x_j après avoir pris la décision $x_i = a$.

Cette agrégation permet de quantifier l'influence d'une variable sur une autre de la manière suivante :

$$I_\alpha(x_i, x_j) = \sum_{v \in D(x_i)} I_\alpha(x_i = v, x_j).$$

La structure des relations entre variables est ainsi représentée par un graphe d'impact associé à chaque mesure α. Ce graphe est défini comme un graphe pondéré orienté $GI_\alpha(X, E, W)$ où X dénote l'ensemble des variables du problème et le poids d'un arc $(x_i, x_j) \in E$ ($E = X \times X$) est donné par $I_\alpha(x_i, x_j)$. Un sommet est donc associé à chaque variable et le poids de chaque arc (x_i, x_j) représente l'influence de x_i sur x_j. Plus le poids est élevé, plus l'influence est grande.

3.4. Illustration des impacts pour l'analyse de structures

Nous cherchons à illustrer à présent l'intérêt du graphe d'impact en analysant des structures sur un cas concret. Nous examinerons notamment comment l'information extraite du graphe d'impact permet à l'utilisateur d'effectuer une analyse du problème ainsi que sa résolution.

3.4.1. *Une instance particulière*

Nous considérons un problème binaire aléatoire dans lequel une structure est insérée en augmentant la dureté de certaines contraintes de manière à faire apparaître plusieurs sous-ensembles de variables entretenant des relations fortes. Les instances aléatoires sont caractérisées par un tuple $\langle N, D, p1, p2 \rangle$ (on utilise le modèle B [1]) où N est le nombre de variables, D la taille des domaines, p_1 la densité du réseau (p_1 est la proportion de contraintes binaires dans le réseau ce qui fixe le nombre de contraintes à $p_1 \times N(N-1)/2$ contraintes) et p_2 la dureté des contraintes (la proportion de couples interdits). Les paramètres de l'instance considérées sont N = 30, D = 10 et $p_1 = 35\%$. Par ailleurs les 30 variables sont divisées en trois sous-ensembles de 10 variables entre lesquelles la dureté est fixé à $p_2 = 53\%$ tandis que le reste du réseau est fixé à 3%.

L'instance spécifique isolée ici à titre d'illustration nous a semblé intéressante à cause de sa difficulté inattendue pour `mindom` [13] (la variable de plus petit domaine courant est choisie en priorité). En utilisant les différents graphes d'impact introduits précédemment, nous souhaitons répondre à différentes questions soulevées face à ce problème :

– est-il possible de reconnaître la structure intégrée au problème ?

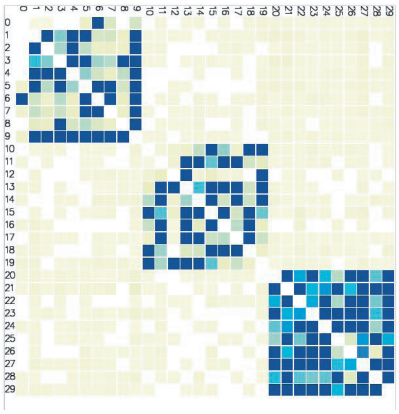

FIGURE 1. Une représentation du graphe d'impact à l'issue de la phase d'initialisation en utilisant la mesure I_0 comme mesure d'impacts.

– pourquoi `mindom` échoue sur cette instance ? Cette difficulté est-elle intrinsèque au problème ou à l'heuristique elle-même ?

3.4.2. *Visualiser le graphe d'impact*

Les figures 1 à 4 donnent une représentation du graphe d'impact GI des 30 variables du problème. Nous utilisons une représentation sous forme de matrice d'adjacence [11]. Les cellules à l'intersection de chaque ligne i et colonne j indiquent l'impact de v_j sur v_i. Plus l'impact est fort, plus l'arc possède un poids élevé et plus la cellule est foncée. La matrice est ordonnée dans l'ordre où sont créés les noyaux cachés de variables[2].

Avant de commencer la recherche, une phase de singleton consistance [22] est appliquée (chaque valeur de chaque variable est propagée) afin d'initialiser l'impact des variables de manière homogène. Bien que le graphe soit entièrement connecté, la visualisation sous forme de matrice donnée en figure 1 permet de distinguer clairement les trois noyaux de variables qui entretiennent des relations fortes, juste après cette première étape de propagation (la mesure I_0 est utilisée ici).

La figure 2 donne une image du graphe d'impact après 2 minutes de recherche en utilisant `mindom` comme choix de variable. Les impacts ne sont donc pas employés ici pour la recherche mais sont maintenus pendant la recherche et permettent d'analyser la résolution effectuée par `mindom`. On peut noter que I_0 est centrée sur les aspects dynamiques de la recherche (les clusters initiaux ne sont plus du tout visibles comparés à la Fig. 1) alors que I_3 est focalisée sur les structures statiques et relègue dans l'ombre les liens faibles même après 2 minutes de calculs (voir Fig. 2). La zone la plus sombre dans le coin en bas à gauche de la figure 2, montre

[2]Nous nous penchons actuellement sur l'utilisation d'algorithmes de *clustering* de manière à identifier cet ordre particulier à partir du graphe d'impact seul.

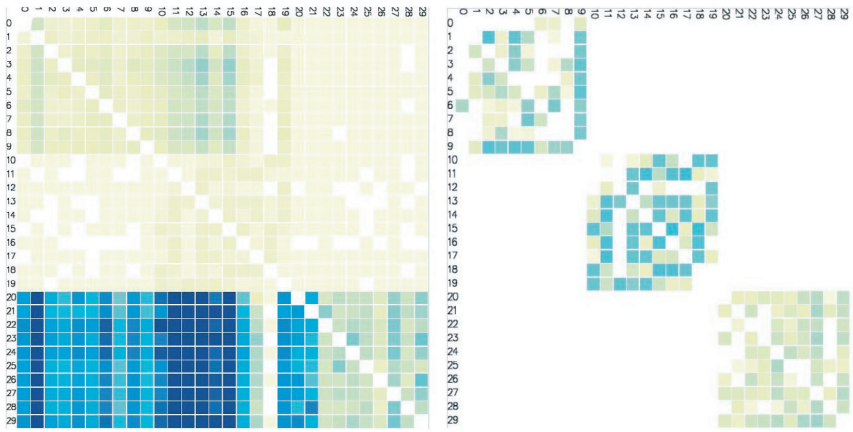

FIGURE 2. Le graphe d'impact basé sur I_0 (graphe de gauche) et I_3 (droite) après deux minutes de calcul en utilisant `mindom`.

que, selon I_0, les variables des deux premiers noyaux ont apparemment une forte influence sur le troisième noyau. La zone deviendrait de plus en plus sombre si la recherche n'avait pas été interrompue à ce stade à des fins d'analyse. On peut penser que de mauvais choix initiaux sur les deux premiers ensembles ont mené le solveur dans une succession d'échecs sur le troisième ensemble masquant ainsi les structures inhérentes au problème.

La figure 3 représente le graphe GI_1 obtenu à partir de la mesure I_1. Il s'agit donc d'un graphe normalisé où l'influence d'une décision prise par le solveur est divisée par le nombre de fois où cette décision est prise pendant la résolution (mesure I_1). L'objectif est de pousser plus loin l'analyse précédente en distinguant deux types de décisions parmi celles qui possèdent une forte influence : certaines sont répétées fréquemment alors que d'autres ont guidé le solveur dans une branche stérile de l'espace de recherche et apparaissent dès lors dans de nombreuses explications d'échecs. On cherche à isoler de cette manière les mauvais choix qui semblent se situer sur le deuxième noyau.

Enfin la figure 4 rapporte l'activité de la résolution en s'appuyant sur un graphe d'impact dans lequel l'effet des anciennes décisions est progressivement éliminé (mesure I_2). Il apparaît alors clairement que le solveur ne cesse d'échouer sur le premier puis plus fréquemment sur le troisième noyau avec très peu d'interaction sur le deuxième où se situent les mauvais choix initiaux. I_2 met en effet l'accent sur les vieilles décisions.

De manière à confirmer cette interprétation, nous avons adapté notre heuristique de recherche pour prendre en compte l'impact des variables au cours de la résolution (en s'appuyant sur I_0) et revenir ainsi immédiatement sur les variables dont l'influence s'accroît de manière indue (parce qu'elles interviennent dans toutes les explications de retrait sans fournir une réduction significative de l'espace de recherche). Le problème est alors résolu instantanément.

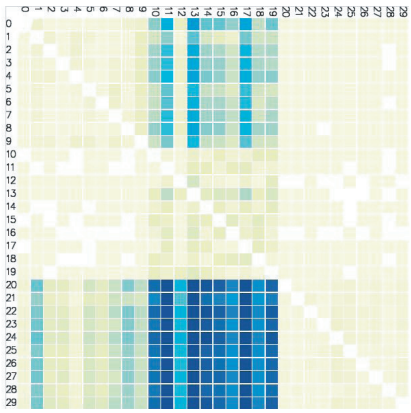

FIGURE 3. Une représentation du graphe d'impact normalisé par rapport au nombre de fois où une décision est prise (I_1).

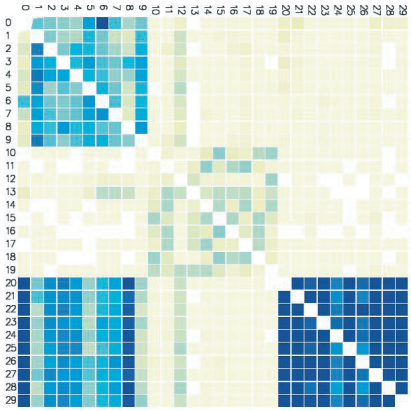

FIGURE 4. Une représentation du graphe d'impact normalisé sur l'âge des décisions (I_2).

4. Les impacts comme heuristiques de recherche

Nous cherchons ici à évaluer si l'impact introduit précédemment constitue une piste sérieuse pour améliorer les heuristiques de recherche génériques.

4.1. Utilisation pour le branchement

Les stratégies de branchement classiques prennent en compte les domaines (`mindom`), comme le degré des variables dans le réseau de contraintes du problème

(`dom + deg` ou `dom/deg`) avec pour objectif d'identifier les variables les plus contraintes, celles qui simplifient au maximum le problème. Les impacts présentés jusqu'à présent dans le cadre de l'identification de structures généralisent de façon naturelle cette idée.

4.1.1. *Impact orienté variables*

Les impacts ont été agrégés jusqu'à présent pour exprimer les relations d'une variable sur une autre. Leur utilisation dans le cadre de la recherche demande une agrégation globale vis à vis de tout le problème. Pour les mesures I_0, I_1 et I_2, l'impact global d'une décision est calculée en accumulant l'impact sur les variables de l'ensemble du problème[3] :

$$\forall \alpha \in [0,1,2], \qquad I_\alpha(x_i = a) = \sum_{x_j \in X} I(x_i = a, x_j).$$

La variable x choisie pour le branchement est celle qui maximise $\sum_{a \in D(x)} I(x = a)$ de manière à favoriser la variable la plus influente ($D(x)$ correspond au domaine courant de x). La valeur choisie est celle qui minimise $I(x = a)$ de manière à favoriser les combinaisons futures consistantes.

4.1.2. *Impact orienté contraintes*

Les impacts ont été définis dans le cadre des contraintes de décisions de manière à porter sur les variables et révéler leurs relations. Ils s'appliquent naturellement sur n'importe quelle contrainte ct de la même manière que I_0 :

$$I(ct) = \sum_{\{e \in E, ct \in e\}} 1/\left|e\right|. \tag{2}$$

L'heuristique reconnue à l'heure actuelle comme la meilleure heuristique standard est sans doute `dom/deg` introduite par [4]. `dom/deg` propose de sélectionner la variable qui minimise le rapport entre la taille de son domaine et son degré dans le réseau de contraintes. Il s'agit de favoriser les petits domaines et les variables les plus contraintes. Elle a été améliorée par [5] (`dom/Wdeg`) en incrémentant le degré d'une contrainte chaque fois que celle-ci lève une contradiction. Nous proposons de raffiner encore davantage cette heuristique en remplaçant le degré par la mesure de l'impact des contraintes portant sur la variable (aboutissant ainsi à l'heuristique nommée par la suite `dom/Ict`).

[3]Le cas d'I_3 est légèrement différent puisqu'il s'agit de se focaliser sur la réduction de l'espace à l'instar de la formule en section 3.3 pour $I_3(x_i = a, x_j)$.

4.2. Experimentations

Pour des raisons de simplicité et de clarté dans les résultats, deux variantes essentielles sont retenues pour tester l'intérêt des impacts pour la recherche : I_2 et `dom/Ict`. Le cadre des expérimentations est le suivant :

- L'utilisation des explications permet de ne pas se limiter à une recherche arborescente classique, mais de remonter directement à l'origine des conflits[4]. Les mesures d'impact sont donc présentées en backjumping (algorithme `mac-cbj` [21]).
- les heuristiques de référence données à titre de comparaison sont `dom/deg`[5] et I_R correspondant à notre implémentation de la mesure introduite dans [23]. I_R consiste à mesurer l'espace de recherche (par le produit cartésien des domaines) avant et après chaque choix pour quantifier précisément la réduction. De manière à séparer l'influence du backjumping, I_R et `dom/deg` sont testés à la fois en mode `mac` et `mac-cbj`. La meilleure combinaison est reportée.
- Les égalités intervenant dans les heuristiques à base d'impact sont arbitrées aléatoirement.
- Enfin, nos expérimentations sont conduites sur un Pentium 4, 3 GigaHz, sous Windows XP. Notre solveur de contraintes est la version java la plus récente du solveur libre `choco` (`choco-solver.net`).

[23] mentionne que l'initialisation des impacts est critique et que l'utilisation de *restart* peut s'avérer parfois payante quand l'initialisation échoue à approximer correctement les impacts. Les impacts deviennent en effet de plus en plus pertinents avec le temps. Les deux techniques sont donc également mises en œuvre ici :

- *restart* : les procédures de *restart* imposent une limite croissante (un doublement à chaque itération garantissant ainsi à terme la complétude) sur le nombre de nœuds autorisés pour la recherche de solution. Les résultats avec *restart* sont indiqués pour une heuristique h (sous la dénomination $h + rest$) dès qu'ils sont meilleurs que h seule.
- *initialisation* : les impacts sont initialisés à l'aide d'une phase de propagation analogue à de la singleton-consistance [22] (chaque valeur de chaque variable est propagée) dont le coût est inclus dans les temps indiqués.

Nous avons considéré quatre ensembles de jeux de tests :

(1) Le premier jeu de tests provient des expérimentations de Refalo [23] : un ensemble de problèmes de multiknapsack modélisés par des variables binaires, et résolus comme un problème de satisfaction (en fixant la valeur optimale comme une contrainte dure). Une limite de temps est fixée à 1500s. De plus, comme les égalités sont arbitrées aléatoirement et que la variance entre deux exécutions est considérable, une moyenne sur une trentaine d'exécutions est reportée.

[4] Il n'y a aucun surcoût à partir du moment où les explications sont maintenues.
[5] `dom/deg` se comporte beaucoup mieux que `mindom` sur nos jeux de tests.

(2) Le deuxième jeu de tests est constitué par un problème binaire aléatoire du modèle B classique (voir Sect. 3.4) avec les paramètres suivants $\langle 50, 10, 30, p_2 \rangle$. On compare ici le nombre d'instance non résolues dans une limite de temps de 120s pour 100 problèmes à chaque valeur de p_2. Une instance est résolue par l'obtention d'une solution ou de la preuve d'inconsistance.
(3) Le troisième ensemble de tests est fait de problèmes binaires aléatoires structurés tels que celui décrit à la Section 3.4. Un problème $\langle 45, 10, 35, p_2 \rangle$ est structuré avec trois noyaux de 15 variables reliés par une dureté intra-noyaux p_2 et une dureté inter-noyaux de 3%. De nouveau, 100 instances sont considérées pour chaque valeur de p_2. On présente sur ces tests les temps de résolution moyens.
(4) Enfin le dernier jeu de tests est issu de problèmes industriels d'allocation de fréquences [6] provenant de l'archive FullRLFAP. Le problème consiste à choisir les fréquences f_i de canaux de communication en minimisant les interférences. Ces contraintes d'interférence s'expriment à travers une distance minimale entre les fréquences des certains canaux ($|f_i - f_j| > E_{ij}$ ou $|f_i - f_j| = E_{ij}$). Nous avons suivi l'approche de [3, 5] pour générer des instances difficiles en satisfaction (il s'agit à l'origine d'un problème d'optimisation). Ainsi, scenXX-wY-fZ correspond à l'instance scenXX dans laquelle les contraintes de poids supérieur à Y ainsi que les Z fréquences les plus hautes ont été éliminées. Les résultats sont donnés sur une sélection de 15 instances difficiles identifiées par [3, 5].

4.3. Premier benchmark : problèmes de multiknapsack

Sur ce premier benchmark (dont les résultats sont indiqués table 1), I_R apparaît comme la meilleure stratégie. On retrouve les résultats de [23] concernant I_R et l'utilisation de *restart* ne profite à aucune des heuristiques testées. L'arbitrage au hasard des égalités prend une trop grande importance pour I_2 et `dom/Ict`[6]. Les deux heuristiques ne sont pas pertinentes et demande un *apprentissage* trop long avant que les impacts ne se stabilisent et permettent de discriminer les variables. Le phénomène s'observe bien pour I_2 en examinant le nombre de nœuds de la dernière itération en cas de *restart* (*cf.* Tab. 1). Ainsi, l'utilisation de *restart* rend I_2 de plus en plus pertinente mais trop lentement et le temps global nécessaire à la résolution augmente.

4.4. Second benchmark : problèmes aléatoires binaires

Sur ce benchmark volontairement non structuré où la taille des domaines (variables entières et non binaires) offre à `dom/deg` de meilleures chances d'évaluer les variables les plus contraintes. Les résultats (visibles sur la Fig. 5 en pourcentage d'instances non résolues – *i.e.* pour lesquelles on a trouvé une solution ou prouvé

[6]Les résultats sont nettement meilleurs sur ce benchmark (pour l'ensemble des heuristiques) avec un arbitrage fixe des égalités.

TABLEAU 1. Impacts sur les problèmes de multiknapsack en arbitrant les égalités au hasard et en indiquant une moyenne sur 30 exécutions.

	mac dom/deg		mac I_R	
	Tps (s)	Nœuds	Tps (s)	Nœuds
mknap1-2	0	11.2	0	24.3
mknap1-3	0	85.9	0	165.7
mknap1-4	0.3	2236.7	0.2	1149.5
mknap1-5	3.6	27749.1	3.5	23158.1
mknap1-6	316.8	2031108.5	201.1	1066116.4

	`dom/Ict`		I_2		I_2 + rest	
	Tps (s)	Nœuds	Tps (s)	Nœuds	Tps (s)	Nœuds
mknap1-2	0	32.8	0	26.0	0	26.0
mknap1-3	0.1	334.5	0.1	594.3	0.1	200.8
mknap1-4	4.3	15063.8	2	7141.5	6	6770.8
mknap1-5	723	2881651.4	234	861328.5	317	446652.6
mknap1-6	> 1500		> 1500		> 1500	

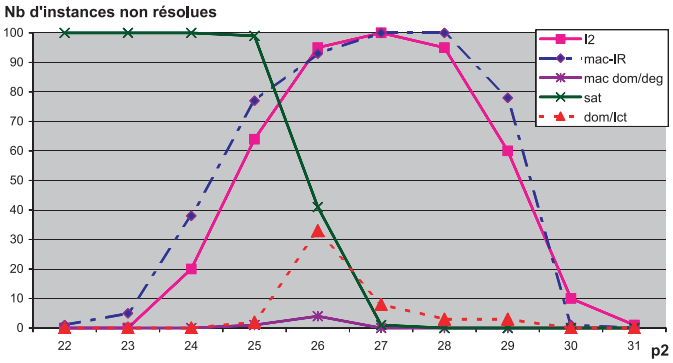

FIGURE 5. Nombre d'instances non résolues et pourcentage d'instances satisfiables.

l'absence), ne sont pas du tout en faveur des impacts. Par ailleurs `mac` surpasse complètement `mac-cbj` et `dom/deg` est aussi bien meilleur que `mindom`.

4.5. TROISIÈME BENCHMARK : PROBLÈMES BINAIRES ALÉATOIRES STRUCTURÉS

Sur ce jeu de test, `mac-cbj` semble critique (`mac dom/deg`, `mac` I_R ou `mac` I_R `+ rest` ne sont pas du tout compétitives). La figure 6 reporte le temps de résolution pour les 4 heuristiques. I_2 comme `dom/Ict` sont incomparablement plus performantes que I_R, elle-même meilleure que `dom/deg`. On peut noter qu'il est inutile de procéder à un *restart* pour I_2 et `dom/Ict` pour qui la phase d'initialisation est suffisante. Ce n'est pas le cas de I_R qui demande plus de temps pour estimer correctement les impacts et que le *restart* améliore considérablement.

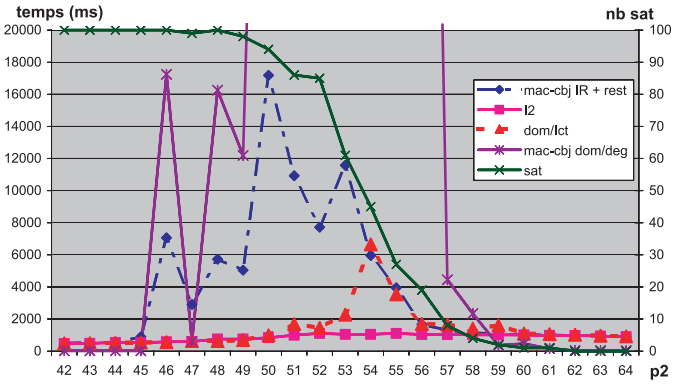

FIGURE 6. Temps de résolution moyens (axe de gauche) et nombre d'instances satisfiables (sat) (axe de droite).

Le succès de I_2 et `dom/Ict` est peut-être dû au fait que la complexité de ce benchmark ne réside pas uniquement au niveau des instances mais aussi certainement au degré élevé d'interaction avec l'algorithme de recherche. La présence de telles structures artificielles favorise de notre point de vue un certain type de comportement *heavy-tailed* [25] qui se caractérise par une grande variation des temps de résolution d'une recherche purement aléatoire indiquant la présence de mauvais point de choix initiaux très difficile à détecter par la suite. On peut ainsi remarquer que le mauvais comportement de I_R se manifeste ainsi juste avant la transition de phase.

4.6. Dernier benchmark : allocation de fréquences

Sur ce benchmark, `mac-cbj` semble décisive comme le souligne les deux premières colonnes du tableau 2 (7 instances résolues pour `mac-cbj` contre 3 pour `mac`). On peut essayer d'analyser ce premier constat à l'éclairage du graphe d'impact[7]. En l'affichant après la phase d'initialisation (première image de la Fig. 7), un utilisateur peut immédiatement constater que le réseau de contraintes est particulièrement peu dense. Par ailleurs, même après une recherche aléatoire (de taille fixe) répétée afin d'exhiber les relations indirectes (seconde image de la Fig. 7), on constate que les impacts ne s'étendent pas à l'ensemble du problème mais restent au contraire assez localisés. Des phénomènes de *thrashing* peuvent facilement apparaître dans ce genre de situation. En effet, comme la propagation reste très locale, une contradiction peut être découverte très tardivement sur une partie du graphe après une longue recherche sur une autre partie (puisque les deux parties entretiennent des relations faibles).

[7]S'agissant par ailleurs d'un problème industriel réel et structuré, il nous a paru intéressant de donner une image du graphe d'impact.

TABLEAU 2. Impacts sur les problèmes d'allocation de fréquences.

		`mac dom/deg`	`mac-cbj dom/deg`	$I_2 + rest$	$I_R + rest$	`dom/Ict`
`scen11`	Temps (s)	47	7.9	38	53	40
(sat)	Nœuds	5863	1207	1432	3986	524
`scen02-f24`	Temps (s)	0.8	0.1	3	3	3
(sat)	Nœuds	620	104	88	90	104
`scen02-f25`	Temps (s)	$> 1h$	3.6	4.6	3.7	8.5
(unsat)	Nœuds	-	610	270	77	1252
`scen03-f10`	Temps (s)	$> 1h$	1766	11.5	9.7	10.5
(sat)	Nœuds	-	527507	1128	415	186
`scen03-f11`	Temps (s)	$> 1h$	$> 1h$	$> 1h$	$> 1h$	17.5
(unsat)	Nœuds	-	-	-	-	788
`scen06-w2`	Temps (s)	$> 1h$	75	14.6	13.5	15.8
(unsat)	Nœuds	-	68669	0	0	0
`scen07-w1-f4`	Temps (s)	0.2	0.2	6	5.9	6.9
(sat)	Nœuds	271	202	194	191	185
`scen07-w1-f5`	Temps (s)	$> 1h$	0	4.4	4.3	5
(unsat)	Nœuds	-	26	0	0	0
`graph08-f10`	Temps (s)	$> 1h$	$> 1h$	$> 1h$	679	19
(sat)	Nœuds	-	-	-	200898	757
`graph08-f11`	Temps (s)	$> 1h$	$> 1h$	$> 1h$	174	14
(unsat)	Nœuds	-	-	-	32653	25
`graph14-f27`	Temps (s)	$> 1h$	$> 1h$	14.9	26.2	32.9
(sat)	Nœuds	-	-	4886	9845	7080
`graph14-f28`	Temps (s)	$> 1h$	$> 1h$	$> 1h$	$> 1h$	14.3
(unsat)	Nœuds	-	-	-	-	1377
nb résolues		3/15	7/15	8/15	10/15	12/15

Les performances des 4 stratégies[8] sont donc présentées au tableau 2 dans leur combinaison avec `mac-cbj`. I_R obtient ici de meilleurs résultats que I_2 qui tous les deux surpassent `dom/deg`. On peut également noter que l'initialisation peut devenir assez coûteuse (autour de 40 s dans le pire des cas) et qu'elle est parfois suffisante à prouver l'inconsistance (instances avec 0 nœuds exploré). `dom/Ict` obtient les meilleurs résultats et permet de résoudre 12 instances sur les 15 faisant partie du benchmark originel surpassant ainsi l'ensemble des autres techniques[9].

4.7. PREMIÈRES CONCLUSIONS SUR LES STRATÉGIES À BASE D'IMPACTS

Les stratégies I_α avec $\alpha \in \{0, 1, 2\}$ sont fortement basées sur l'activité du solveur pendant la recherche (en se focalisant donc sur la partie dynamique des structures). Leur utilisation peut s'avérer extrêmement rentable dans la mesure où elles révèlent les mauvais choix initiaux (dont l'influence grandit démesurément au cours de la

[8]Le produit cartésien des domaines nécessaire au calcul d'I_R conduit ici à des débordements d'entier. I_R ne devient donc discriminante qu'à partir du moment où l'espace est suffisamment réduit. Pour contourner ce problème et ne pas pénaliser I_R, nous avons basé la mesure de la réduction de l'espace de recherche sur le nombre de valeurs retirées au lieu du produit cartésien.

[9]Les instances `scen11-f1`, `scen11-f2`, `scen11-f3` n'ont pu être résolues dans la limite de temps d'une heure par aucun des algorithmes.

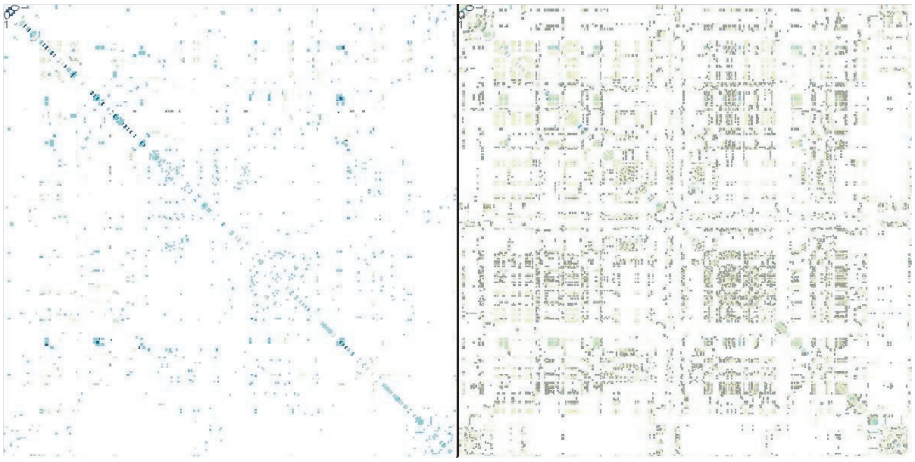

FIGURE 7. Le graphe d'impact des variables après l'initialisation et après une recherche aléatoire répétée pour essayer de construire une vue homogène des relations.

recherche sans apporter un élagage utile puisque le solveur ne parvient pas à revenir dessus). Par ailleurs, quand l'impact est appliqué aux contraintes, il révèle celles qui sont responsables à court ou long terme de la propagation. En revanche, nous pensons que l'impact basé sur les explications varie trop d'un nœud à l'autre de l'arbre de recherche (précisément parce qu'il se focalise sur les aspects dynamiques) pour faire des I_α à elles seules des heuristiques génériques robustes.

I_3 est trop coûteuse (en temps) dans l'état actuel pour être utilisée comme heuristique par défaut mais d'autres compromis intéressants entre I_R et I_α avec $\alpha \in \{0, 1, 2\}$ peuvent être imaginés de manière à profiter de la robustesse générale de I_R tout en essayant de se prémunir contre des phénomènes de *thrashing*[10] liés au choix initiaux.

Les expérimentations reportées ici ont cherché à affiner notre compréhension des phénomènes mis en jeu pendant la recherche et ont mis en évidence l'intérêt pour I_R de s'appuyer aussi sur une information plus fine provenant du calcul des explications. Leur pertinence est ainsi mise en avant sur le dernier benchmark qui a l'avantage d'être issu d'une problématique réelle.

5. Exploitation de structures spécifiques

La décomposition de Benders en Recherche Opérationnelle est une technique dédiée à des problèmes qui possèdent une structure statique présentant un sous ensemble de variables avec un impact important sur l'ensemble du problème et où, idéalement, le reste du problème se décompose en sous-problèmes indépendants. La

[10]Le thrashing est l'exploration redondante et inutile de régions de l'espace de recherche dont l'inconsistance n'est pas lié aux décisions remises en question

décomposition utilise une relation maître-esclave entre variables. Les variables du maître sont appelées *variables compliquantes* dès 1972 par Geoffrion [10]. Une fois ces variables instanciées, le sous-problème d'optimisation résultant est beaucoup plus simple.

On souhaite mettre en œuvre ce type de décomposition, qui a largement démontré son efficacité en Recherche Opérationnelle, dans le contexte de structures *cachées* et révélées par le graphe d'impact (sous la forme de sous ensembles ayant de fortes intra-relations et de faibles inter-relations). La mise en œuvre d'une telle technique en programmation par contraintes n'est néanmoins pas immédiate. Traditionnellement, les coupes de Benders sont limitées à la programmation linéaire et calculée par la résolution du dual du sous-problème[11]. Elles exigent donc que les variables duales puissent être définies pour appliquer la décomposition. Cependant, Hooker et Ottosson [14] proposent de dépasser cette limite et d'élargir le concept de dualité en introduisant un *dual logique* valable pour tout type de sous-problème. Ils se réfèrent à un schéma plus général et suggèrent une manière différente de penser la dualité en s'appuyant sur la logique, la capacité de produire une preuve et un ensemble d'hypothèses suffisantes pour cette preuve.

Néanmoins, ce dual logique demande à être implémenté pour chaque classe de sous-problèmes en identifiant les coupes proprement dites [7, 15]. On peut considérer le dual comme un certificat d'optimalité, autrement dit, une explication (comme introduit en Sect. 3.1) d'inconsistance dans notre cas. Le paradigme de la programmation par contraintes expliquée fournit d'une certaine manière une implémentation générique de la décomposition de Benders généralisée dans le cas de problèmes de satisfaction [7]. On peut noter ici que le calcul des explications est paresseux[12], la première explication trouvée est conservée alors qu'elle n'est pas unique. Le calcul de l'explication minimale à la volée pendant la résolution est évidemment trop coûteux. Par conséquent, un tel dual logique fournit une solution duale arbitraire[13] mais pas nécessairement la solution optimale. À l'évidence, le succès d'une telle approche dépend du degré avec lequel des explications précises peuvent être calculées pour les contraintes du sous-problème.

La programmation par contraintes à base d'explications propose des algorithmes du type de `mac-dbt` [18] ou `decision-repair` [19] qui essayent de se focaliser naturellement sur le problème maître d'une telle décomposition mais peuvent revenir à un comportement plus conventionnel. L'étape suivante consiste à utiliser la structure exhibée par le graphe d'impacts pour appliquer un schéma de décomposition dans une seconde phase de résolution. L'identification de sous-problèmes pourrait par ailleurs guider la génération de coupes pour le problème maître de manière à rassembler autant d'information que possible là où réside la vraie combinatoire du problème.

[11] En référence à la dualité dans un contexte linéaire.

[12] L'ensemble des explications n'est pas calculé suite à un retrait de valeur. Seule l'explication correspondant au raisonnement courant du solveur est conservée.

[13] Ce qui est également vrai en programmation linéaire où toute solution duale constitue une borne pour le primal.

6. Conclusion

Nous avons introduit dans cet article plusieurs indicateurs qui peuvent s'avérer utiles à la fois pour l'identification et l'utilisation de certaines structures clef au cœur d'un problème combinatoire. Nous nous sommes intéressés dans cette étude aux relations entre les variables en ouvrant également de nouvelles perspectives sur la conception d'heuristiques de recherche génériques pour la programmation par contraintes comme la conception de nouveaux algorithmes qui pourraient être proposés par les solveurs. Nous pensons que la présence de *backdoors* ou de sous-ensembles de variables présentant un impact important sur l'ensemble du problème pourrait être utilisé explicitement par des stratégies de décomposition *ad hoc* inspirées de la Recherche Opérationnelle. La décomposition de Benders et son extension logique en est un exemple. Il s'agit en effet d'une technique de *backdoors* qui pourrait être appliquée en programmation par contraintes comme une stratégie d'enregistrement de *nogood*.

Références

[1] D. Achlioptas, L. Kirousis, E. Kranakis, D. Krizanc, M. Molloy and Y. Stamatiou, Random constraint satisfaction : a more accurate picture, in *Proceedings CP 1997*, Linz, Austria (1997) 121–135.

[2] J.F. Benders, Partitionning procedures for solving mixed-variables programming problems. *Numer. Math.* **4** (1962) 238–252.

[3] C. Bessière, A. Chmeiss and L. Saïs. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem, in *Proceeding CP'01*, Paphos, Cyprus (2001) 565–569. Short paper.

[4] C. Bessiere and J.C. Regin, MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems, in *Proceeding CP'96* (1996) 61–75

[5] F. Boussemart, F. Hemery, C. Lecoutre and L. Sais, Boosting systematic search by weighting constraints, in *Proceedings ECAI'04* (2004) 482–486.

[6] B. Cabon, S. de Givry, L. Lobjois, T. Schiex and J.P. Warners, Radio Link Frequency Assignment. *Constraints* **4** (1999) 79–89.

[7] H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien and Y. Trinquet, Decomposition and learning for a real time task allocation problem, in *Proceedings CP 2004* (2004) 153–167.

[8] H. Cambazard and N. Jussien, Integrating Benders decomposition within Constraint Programming, in *Proceedings CP 2005*, short paper (2005) 752–756.

[9] G. Cleuziou, L. Martin and C. Vrain, Disjunctive learning with a soft-clustering method, in *ILP'03 :13th International Conference on Inductive Logic Programming*, LNCS, September (2003) 75–92.

[10] A.M. Geoffrion, Generalized Benders Decomposition. *J. Optim. Theory Practice* **10** (1972) 237–260.

[11] M. Ghoniem, N. Jussien and J.-D. Fekete, VISEXP : visualizing constraint solver dynamics using explanations, in *Proceedings FLAIRS'04*, Miami, Florida, USA, May (2004) 263–268.

[12] C.P. Gomes, B.t Selman and N. Crato, Heavy-tailed distributions in combinatorial search, in *Proceeding CP 97*, Linz, Austria (1997) 121–135.

[13] R. Haralick and G. Elliot, Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence* **14** (1980) 263–313.

[14] J.N. Hooker and G. Ottosson, Logic-based benders decomposition. *Math. Program.* **96** (2003) 33–60.

[15] V. Jain and I.E. Grossmann, Algorithms for hybrid MILP/CP models for a class of optimization problems. *Informs J. Comput.* **13** (2001) 258–276.
[16] N. Jussien, *The versatility of using explanations within constraint programming*. Habilitation thesis, Université de Nantes, France, also available as RR-03-04 research report at École des Mines de Nantes (2003).
[17] N. Jussien and V. Barichard, The PaLM system : explanation-based constraint programming, in *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Singapore (2000) 118–133.
[18] N. Jussien, R. Debruyne and P. Boizumault, Maintaining arc-consistency within dynamic backtracking, in *Proceedings CP 2000*, Springer-Verlag, Singapore (2000) 249–261.
[19] N. Jussien and O. Lhomme, Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence* **139** (2002) 21–45.
[20] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman and L. Troyanski, Determining computational complexity for characteristic 'phase transitions', in *Nature* **400** (1999) 133–137.
[21] P. Prosser, *MAC-CBJ : maintaining arc-consistency with conflict-directed backjumping*. Research report 95/177, Department of Computer Science – University of Strathclyde (2005).
[22] P. Prosser, K. Stergiou and T. Walsh, Singleton consistencies, in *Proceedings CP 2000*, edited by R. Dechter, Singapore (2000) 353–368.
[23] P. Refalo, Impact-based search strategies for constraint programming, in *Proceedings CP 2004*, Toronto, Canada (2004) 556–571.
[24] J.-C. Régin, A filtering algorithm for constraints of difference in CSPs, in *AAAI 94, Twelth National Conference on Artificial Intelligence*, Seattle, Washington (1994) 362–367.
[25] R. Williams, C. Gomes and B. Selman, On the connections between backdoors and heavy-tails on combinatorial search, in *the International Conference on Theory and Applications of Satisfiability Testing (SAT)* (2003).
[26] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity, in *Proceedings IJCAI 2003* (2003).